OpenMP Raport

Zuzanna Rękawek

Kwiecień 2021

1 Specyfikacja

- Procesory logiczne: 4
- Procesory logiczne: 8
- OS: Windows 10
- Typ systemu: x64

Zadanie zostało wykonane przy użyciu środowiska Microsoft Visual Studio 2019. Wszystkie programy zostały skompilowane z konfiguracją Release, a następnie uruchomione bez debbugowania.

2 Przetwarzanie sekwencyjne

Czas wykonywania obliczeń: 0.196[s]

3 Pomiary czasu

Poniższe wartości prezentują długość przetwarzania dla kolejnych wersji kodu w sekundach- podstawowej jednostce czasu układu SI.
Wyniki zostały zmierzone dzięki funkcji *omp_get_wtime*() z biblioteki OMP.

Zad1	Wątki			Wynik
	8	4	2	
PI2	0.247108	0.168973	0.157797	niepoprawny
PI3	10.911656	7.476831	4.132476	poprawny
PI4	0.080543	0.109149	0.115842	poprawny
PI5	0.047536	0.061145	0.114769	poprawny
PI6	0.061080	0.084447	0.105977	poprawny

Tablica 1: czasy przetwarzania

4 Omówienie działania programów

4.1 PI2

Wprowadzając derektywę *#pragma omp paralel for* sprawiono, że zmienna i jest prywatną. Pozostałe zmienne są współdzielone.
Znaczącym problemem jest współdzielenie zmiennych x oraz *sum*- są dostępne do odczytu i zapisu. Prywatność zmiennej x możemy uzyskać dzięki każdorazowej deklaracji zmiennej w pętli. Współdzielenie jej jest niepoprawne- każdy wątek oblicza własną wartość.
Wieloktrotny zapis i odczyt zmiennej *sum* powoduje wyścig- możliwość niesynchronizowanych dostępów. Zapis powoduje unieważnienie kopii linii zawierającej tą zmienną w innym procesorze, czego następstwem jest nieporpawny wynik końcowy.
Przewidzane przyspieszenie nie występuje w każdym przypadku, ze względu na powyższe problemy z poprawnoscią dostępu do zmiennych.

4.2 PI3

W tym przykadzie lokalność zmiennych nie ulega zmianie. Upewniono się, ze zmienna x jest prywatna dla każdego wątku.
Efektem użycia dyrektywy *#pragma omp atomic* jest otrzymanie dobrego wyniku. Czas wykonywania obliczeń uległ pogorszeniu. Jest to spowodowane charakterystyką dyrektywy.
Dyrektywa *#pragma omp atomic* wymusza niepodzielnośc podczas odczu oraz zapisu zmiennej *sum*, będącą zmienną współdzielną przez wszystkie wątki, jest wykonywane sekwencyjnie.
Zapewnia również synchronizację na poziomie sprzętowym, czego konsekwencją jest unieważnienie linni pamięci na wszystkich innych procesorach oraz pamięci operacyjnej, które zawierają tą zmienną. Synchronizacja wątków jest realizowana za pomocą zakładanego zamka.

4.3 PI4

W celu unieknięcia wieloktornego zapisu i odczytu zmiennej *sum* uzywajac dyrektywy *#pragma omp atomic*, wprowadzono zmienną prywatne *sum*1, przechowującą sumy dla poszczególnych wątków.
Dyrektywa *#pragma omp atomic* w dalszym ciągu jest używana przy końcowym zapisie do zmiennej- w tym przypadku odbywa się to poza pętlą, więć kazdy wątek robi to tylko raz na sam koniec. Lokalnosc innych zmiennych nie uległa zmienie.

4.4 PI5

Wprowadzneie klauzuli *reduction*$(+ : sum)$ dodanej do *#pragma omp for* zapewnia prywartność zmiennej *sum*, mimo jej deklaracji poza pętlą- za każdym

razem tworzona nowa, prywatna zmienna.
Pisząc *reduction*(: *sum*) informujemy kompilator, że do zmiennej *sum* będziemy dodawać kolejno wyliczone wartości właśnie tej zmiennej. Kompilator utworzy odpowiednią liczbę prywatnych kopii zmiennej dla każdego wątku i rozdzieli iteracje pomiędzy dostępne wątki. Każdy wątek będzie operował tylko na swojej kopii zmiennej *sum*. Po wykonaniu wszystkich iteracji, wartości wyliczone przez wszystkie wątki są do siebie dodawane. Obliczona wartość zmiennej *sum* typu reduction jest dostępna poza sekcją parallel.
Zastosowanie klauzuli jest równoważne z operacjami w PI4. Czasy przetważania są bardzo podobne, biorąc pod uwagę ich szybkośćwykoanania.
Trzeba pamiętać, że obecne rozwiązanie ma swoje wady- przy większych, pod względem zajmowanej pamięci, strukturach może zdarzyć się, że obiekt nie mieści się na pamieci poziomu pierwszego. Wątek traci czas na ubieganie się o miejsce na zapis zmiennej, co powoduje dłuższe przetwarzanie.

4.5 PI6

W tym przykładzie zasowano tablicę jako strukturę przechowująca zmienne prywatne, wywoływane unikalnymi numerami id danego wątku.
Problem z odwoływaniem się do tego samego słowa

4.6 PI7

Celem eksperymentu było wyznaczenie długości linii pamięci poprzez znalezienie adresów odpowiadających początkom i końcom linii. Aby to osiągnąć należało ustawić dwa wątki tak, by pracowały na sąsiednich elementach tablicy: 0 i 1, 1 i 2, ... Dzięki temu możliwym do zaobserwowania był moment, gdy jeden z watków pracował wciąż na jednej lini pamięci, a drugi przechodził już na kolejną: następowało wtedy przyspieszenie działania z powodu braku false sharingu. Wyznaczenie linii pamięci można obliczyć z częstotliwości pojawiania się przyspieszeń, w tym przypadku 8 * 8 (8=sizeof(double)) = 64B. Taka długość linii pamięci podręcznej procesora jest zgodna z architekturą komputera (x64).